تمرین کامپیوتری سوم

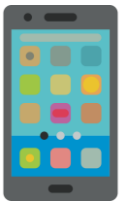
دانشکده مهندسی برق و کامپیوتر | گزارش کار | سیستم‌های عامل - پاییز ۱۳۹۹

استاد:
دکتر مهدی کارگهی

تاریخ: امروز

نام و نام خانوادگی:
احسان اسکندری
810197654

مقدمه



در این تمرین به تحلیل داده هایی که از مشخصات و قیمت فروش گوشی‌های موبایل جمع‌آوری شده‌است پرداخته شده است. در ابتدا برنامه اقدام به خواندن و تجزیه مجموعه داده[1]ی ارائه شده می‌کند و آنها را در حافظه خود ذخیره می‌کند. پس از استخراج داده‌ها و ویژگی‌های آنها، برنامه اقدام به نرمال‌سازی[2] داده‌ها و در نهایت اقدام به تعیین طبقه قیمتی گوشی‌ها می‌کند. این تمرین به دو روش این مسئله پیاده‌سازی شده است که در ادامه گزارش، نتایج حاصل آمده است.

[1] Dataset

[2] Data Normalization

پیاده‌سازی سری

سوال اوّل

چرا برای پیاده‌سازی یک برنامه بصورت چندریسه‌ای، بهتر است ابتدا این برنامه بصورت سری پیاده‌سازی شود؟

ابتدا بهتر است از درستی کلی برنامه مطمئن شویم تا راه حل کلی را بدانیم و ایراد های مربوط به پروژه را راحت تر بفهمیم تا اینکه وارد پیچیدگی های چند ریسه ای شویم.
همچنین بهتر است ابتدا راه حل سری را بررسی کنیم تا متوجه شویم بهتر است کدام قسمت ها را چند ریسه ای اجرا کنیم و بیشترین زمان را مصرف می کند.

سوال دوم

با بررسی زمان اجرای بخش‌های مختلف برنامه، **Hotspot**[3] های برنامه را مشخص کنید.

دو عکس بعدی، اجرای برنامه با ورودی داده شده و ورودی 1 میلیون تایی(همان 2000تا تکرار شده اند) را نشان می دهد. بعد از هر قسمت برنامه، زمان گرفته شده است تا متوجه شویم چه درصدی از برنامه در کدام توابع مصرف می شود.

[3] توابعی که در برنامه‌تان بیشترین زمان اجراها را به خود اختصاص می‌دهند.

```
ehsan@ubuntu:~/Desktop/OS3/serial$ make && time ./a.out heavy_dataset/
g++ -std=c++11 -I  -I  -pthread -c a.cpp -o a.o
g++ -std=c++11 -I  -I  -pthread a.o -o a.out
Accuracy: 93.05%

Total Elapsed Time: 28306 ms

          Read Dataset  16704ms     59.01%
     Normalize Dataset   6507ms     22.99%
      Calculate Result   5076ms     17.93%
   Calculate Precision     18ms      0.06%

real    0m28.602s
user    0m27.819s
sys     0m0.781s
```

```
ehsan@ubuntu:~/Desktop/OS3/serial$ make && time ./a.out dataset/
make: Nothing to be done for 'all'.
Accuracy: 93.05%

Total Elapsed Time: 67 ms

          Read Dataset     43ms     64.34%
     Normalize Dataset     16ms     23.95%
      Calculate Result      7ms     11.67%
   Calculate Precision      0ms      0.05%

real    0m0.074s
user    0m0.069s
sys     0m0.005s
```

همانطور که میبینید، بیشترین زمان برنامه برای خواندن دیتاست و نرمالایز کردن آن میگذرد.

جدول اوّل

زمان‌های اجرای ۶ اجرای متوالی از برنامه و میانگین آن‌ها را بازای ورودی نمونه‌ای که در شرح تمرین آمده است، در جدول زیر بیاورید.

اجرای اوّل	اجرای دوم	اجرای سوم	اجرای چهارم	اجرای پنجم	اجرای ششم	میانگین
67ms	55ms	61ms	59ms	57ms	60ms	60ms

سوال سوم

اگر هنگام موازی‌سازی برنامه به زمان اجرای بیشتری نسبت به حالت سری برخورد کنید، چه رویکردهایی را برای کاهش زمان اجرا و استفاده حداکثری از موازی‌سازی پیش می‌گیرید؟

تعداد **lock** های استفاده شده و مدت زمانی که هر ترد ممکن است به حالت **stall** بر بخورد، را کاهش میدهیم تا **overhead** ناشی از آن بیشتر از زمان بهبود یافته نشود. برای اینکار ترد ها را تا حد ممکن از هم مستقل می کنیم تا با حداکثر توان پردازش انجام دهند.

سوال چهارم

در هنگام پیاده‌سازی این بخش، به چه چالش‌هایی برخورد کردید و بیان کنید که به چه صورت آن‌ها را رفع کردید.

اولین چالش، ایجاد mutex ها و lock های لازم بود که در قسمتی از شناسایی critical section اشتباه کرده بودم و مدت زمانی رو مشغول درست کردن آن بودم. بعد متوجه شدم در آن واحد همواره فقط یک ریسه مشغول خواندن دیتاست است و این قسمت تفاوتی با حالت یک ریسه نمی کند. به همین خاطر دیتاست را به 5 قسمت مساوی تقسیم کردم تا هر قسمت برای خودش عملیات خواندن را انجام دهد و در نهایت همه اینها را برای قسمت بعد تجمیع کنم. که به کمک این سرعت این قسمت کمی بهتر شد.

با توجه به اینکه کد سری را تدریجی به موازی تبدیل می کردم، قسمت بعدی فقط یک vector از دیتاست لازم داشت و مجبور به تجمیع بودم که بعد متوجه شدم قسمت اعظمی از قسمت بعد را نیز می توان موازی انجام داد. بنابراین تصمیم گرفتم دیتاست ها را در 5 وکتور مختلف نگهداری کنم.

در قسمت نرمالایز کردن دیتا، باید min و max کل دیتاست برای هر ویژگی را میداشتیم و ریسه ها به هم وابستگی داشتند، به همین خاطر برای حل این مشکل، هر ریسه min وmax دیتا خود را انجام میداد و سپس min و max های کل دیتاست بر اساس دیتا خود انجام میداد و سپس منتظر بقیه ریسه ها بود.

با این روش، بعد از بدست آوردن min و max دیگر وابستگی نبود و ریسه ها میتوانستند کار خود را بر دیتاست خود انجام دهند.

در نهایت، جواب های هر ریسه با هم جمع شده و ریسه اصلی جواب نهایی را محاسبه و چاپ می کند.

سوال پنجم

با توجه به تجربه‌ای که در پیاده‌سازی این تمرین بدست آوردید، به نظر شما در چه مواقعی از قفل[4] در یک طراحی چندریسه‌ای ضروری است؟ تاثیر استفاده از قفل‌ها را بر روی کارآیی[5] سامانه بیان کنید.

زمانی که ریسه های مختلف منابع مشترکی داشته باشند و بخواهند همزمان آن را تغییر دهند.

به دلیل استفاده از قفلها، یک ریسه مجبور به stall می شود تا کار دیگر ریسه تمام شود که باعث می شود کارایی برنامه کاهش یابد. همچنین قفل کردن و بازکردن و چگ کردن قفل نیز زمانی صرف میشود که overhead بیشتری ایجاد می کند و کارایی سامانه را نیز کاهش میدهد.

[4] Lock

[5] Performance

جدول دوم

زمان‌های اجرای ۶ اجرای متوالی از برنامه و میانگین آن‌ها را بازای ورودی نمونه‌ای که در شرح تمرین آمده است، در جدول زیر بیاورید.

دو عکس زیر، از اجرای چندریسه ای روی همان ورودی صحبت شده در قسمت اول است. همانطور که میبینید کاهش محسوسی(مخصوصا در دیتای سنگین) در زمان اجرای برنامه داشتیم.

```
ehsan@ubuntu:~/Desktop/OS3/parallel$ make && ./PhonePricePrediction.out dataset5t/
make: Nothing to be done for 'all'.
Accuracy: 93.05%

Total Elapsed Time: 37 ms

          Read Dataset      14ms      39.92%
     Normalize Dataset      14ms      38.83%
      Calculate Result       7ms      19.60%
   Calculate Precision       0ms       1.66%
```

```
ehsan@ubuntu:~/Desktop/OS3/parallel$ make && ./PhonePricePrediction.out heavy_dataset/
make: Nothing to be done for 'all'.
Accuracy: 93.05%

Total Elapsed Time: 14509 ms

          Read Dataset    6734ms      46.41%
     Normalize Dataset    6487ms      44.71%
      Calculate Result    1281ms       8.83%
   Calculate Precision       6ms       0.04%
```

اجرای اوّل	اجرای دوم	اجرای سوم	اجرای چهارم	اجرای پنجم	اجرای ششم	میانگین
37ms	40ms	38ms	38ms	40ms	43ms	39ms

میزان تسریع ($\frac{Serial\ Time}{Parallel\ Time}$) برنامه نسبت به حالت سری را در زیر بیاورید.

میانگین زمان اجرای سری	میانگین زمان اجرای موازی	میزان تسریع
60ms	39ms	1.53

در حالت دیتاست 1 میلیون تایی، تسریع دوبرابر داشتیم.